Mathematics and Computing/Technology
MT365 Graphs, networks and design

MT365 ATN2

The Open University

MT365
Audio Notes 2

CDA 5675 from TRACK 9 pp 4 – 19
CDA 5676 pp 20 – end

3.4

SUP 54568 7

This booklet contains the printed material for use with Audio-tape 3 for the audio-tape sections of *Networks 3* and Audio-tape 4 for the audio-tape sections of *Graphs 4*.

You will need to play the tape at the same time as you study the frames on the following pages.

Place the cassette player within easy reach. There are points on the tape where we have indicated that we want you to stop the tape and do some work for yourself, but you will probably find it necessary to stop the tape more often than this. Indeed, you should get into the habit of frequently stopping the tape and giving yourself time to think.

Make sure that you have paper and pencil handy before starting each tape sequence.

BLOCK 3

Notes for Networks 3

There are three sequences on the tape for this unit. The heading numbers below refer to the corresponding sections in *Networks 3*.

In each tape sequence we demonstrate the use of an algorithm to solve an example. In the first two tape sequences we then ask you to use the algorithm to solve a problem. There is a problem on the third algorithm in the text of the unit. Additional problems requiring the use of these three algorithms are given in the *Computer Activities Booklet*.

Each algorithm involves finding matchings in bipartite graphs, and is based on the idea of an *alternating path*, introduced in Section 2. As a reminder, this definition is given below.

Definition

Let G be a bipartite graph in which the set of vertices is divided into two disjoint subsets X and Y. An **alternating path** with respect to a matching M in G is a path which satisfies the conditions:

(a) the path joins a vertex x in X to a vertex y in Y;

(b) the initial and final vertices x and y are not incident with an edge in M;

(c) alternate edges of the path are in M, and the other edges are not in M.

2.2 Maximum matching algorithm

3.1 Hungarian algorithm for the assignment problem

4.1 Hungarian algorithm for the transportation problem

A formal statement of each algorithm is given in the unit. **You should have the unit open at the appropriate page whilst listening to each tape sequence.**

Page 14 has been left blank to enable other frames to face each other.

BLOCK 4

Notes for Graphs 4

There are two sequences on the tape, both associated with Section 5. The numbers 5.3 and 5.4 below refer to the corresponding sections in *Graphs 4*.

In each sequence we describe an algorithm for solving a particular problem using a branch and bound method. The problems are:

5.3 the knapsack problem;

5.4 the travelling salesman problem.

Each algorithm involves a search for an optimum solution based on the following ideas:

- a branching tree for structuring the search;
- successive improvement of a lower bound for a number to be determined.

In the case of the knapsack problem, the number to be determined is the *total value of items packed*.

In the case of the travelling salesman problem, the number to be determined is the *total length of the route*.

TRACK9

The maximum matching algorithm

TRACK 10

1 WORKED PROBLEM

Find a maximum matching in the following bipartite graph.

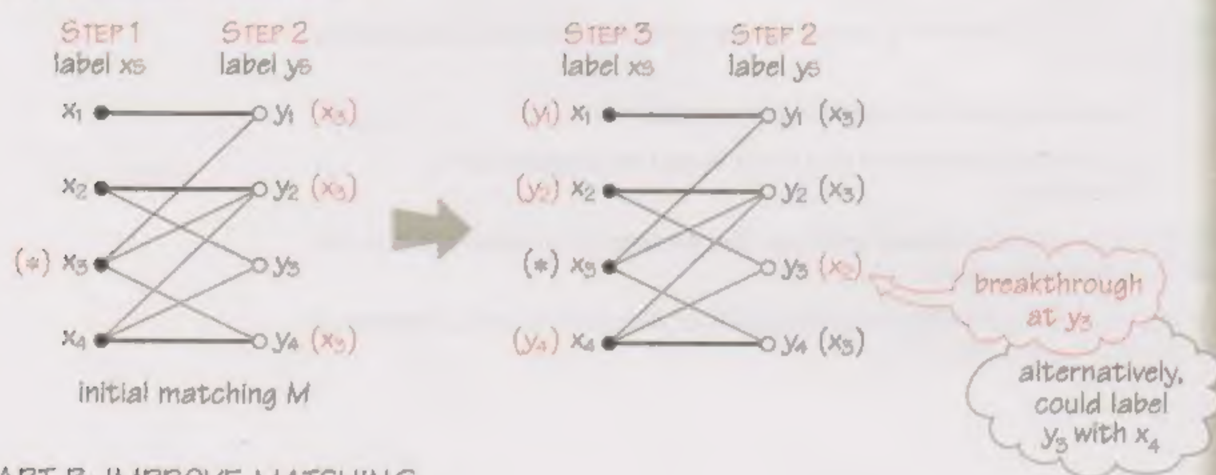
USE

- labelling procedure
- matching improvement procedure

TRACK 11

2 SOLUTION TO WORKED PROBLEM

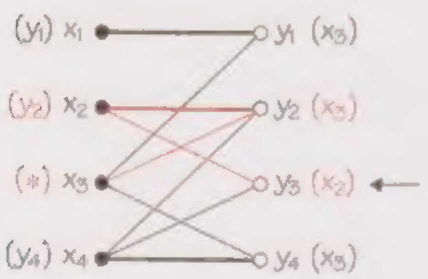
PART A: LABEL VERTICES



initial matching M

PART B: IMPROVE MATCHING

STEP 4: FIND ALTERNATING PATH



an alternating path is $y_3x_2y_2x_3$

alternatively: $y_3x_4y_4x_3$

2 SOLUTION CONTINUED

STEP 5: CONSTRUCT REVISED MATCHING

original matching
x_1y_1, x_2y_2, x_4y_4

revised matching
x_1y_1, x_4y_4, x_2y_3, x_3y_2

alternatively:
x_1y_1, x_2y_2, x_3y_4, x_4y_3

Since the revised matching has 4 edges, it is a maximum matching.

TRACK 12

3 SUMMARY OF THE ALGORITHM

START with any matching.

Part A: labelling procedure

Label the vertices to identify an alternating path.
If *breakthrough* is achieved, go to Part B.
If *breakthrough* is not achieved, STOP:
the current matching is a maximum matching.

breakthrough occurs when a vertex in Y not incident with any edge in the current matching is labelled

Part B: matching improvement procedure

Find an alternating path by tracing back through the labels.

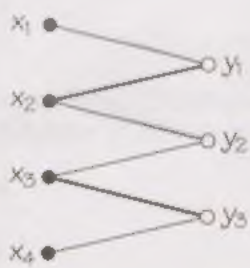
Form a new matching from:
- the edges in the current matching NOT IN the alternating path,
- the edges in the alternating path NOT IN the current matching.

Return to Part A.

TRACK 13

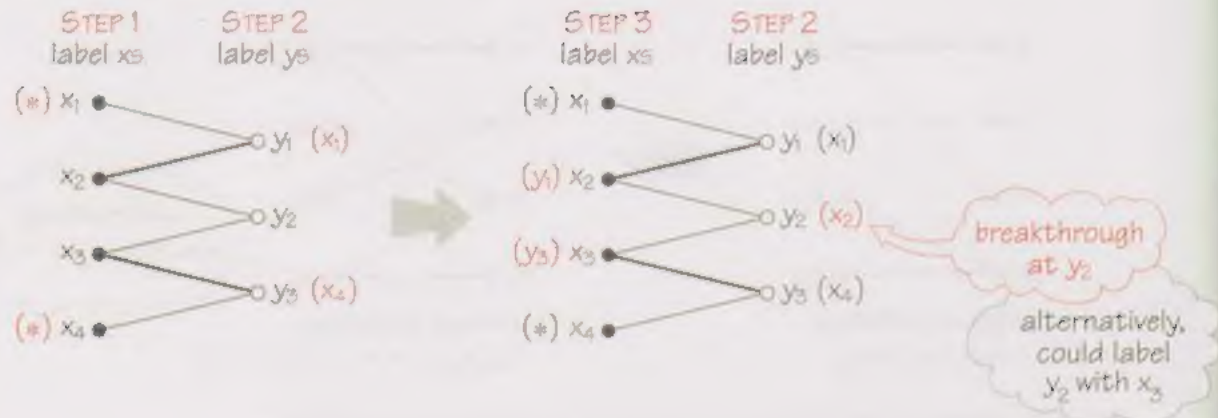
4 PROBLEM

Find an improved matching in the following bipartite graph.



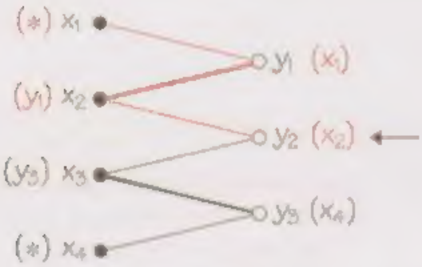
5 SOLUTION

PART A: LABEL VERTICES



PART B: IMPROVE MATCHING

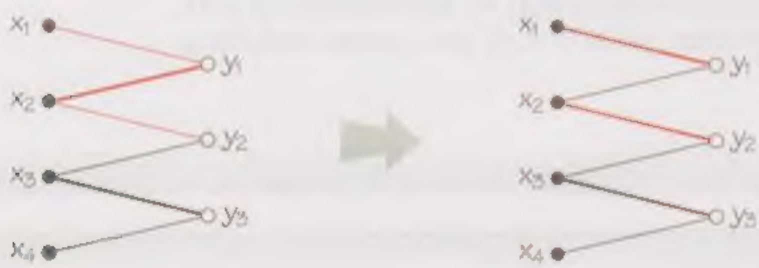
STEP 4: FIND ALTERNATING PATH



an alternating path is $y_2x_2y_1x_1$

alternatively: $y_2x_3y_3x_4$

STEP 5: CONSTRUCT REVISED MATCHING



initial matching
x_2y_1, x_3y_3

revised matching
x_3y_3, x_1y_1, x_2y_2

alternatively:
x_2y_1, x_3y_2, x_4y_3

Since the revised matching has 3 edges and there are only 3 vertices in Y, it is a maximum matching.

TRACK 15

The Hungarian algorithm for the assignment problem

USE
- labelling procedure
- matching improvement procedure
- modification of partial graph procedure

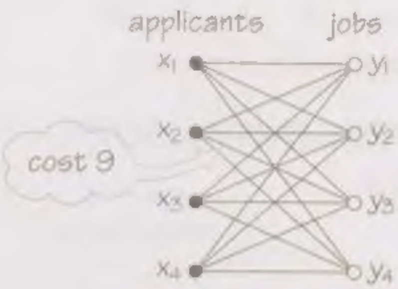
1 WORKED PROBLEM

Find the optimum assignment in the following case.

jobs

applicants		y_1	y_2	y_3	y_4
	x_1	6	12	15	15
	x_2	4	8	9	11
	x_3	10	5	7	8
	x_4	12	10	6	9

cost matrix



bipartite graph $K_{4,4}$

TRACK 16

2 SOLUTION TO WORKED PROBLEM

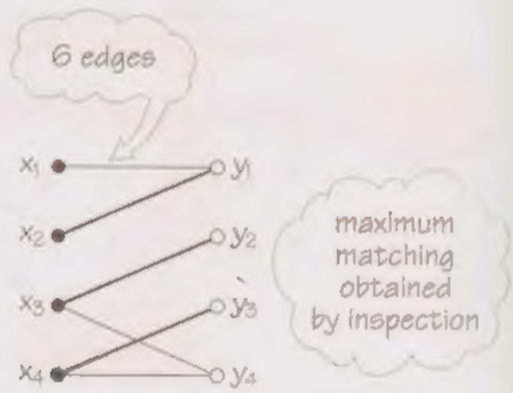
STEP 0: CONSTRUCT INITIAL PARTIAL GRAPH

weights ↓		y_1	y_2	y_3	y_4
6	x_1	0	6	9	9
4	x_2	0	4	5	7
5	x_3	5	0	2	3
6	x_4	6	4	0	3

weights →		0	0	0	3
weights ↓		y_1	y_2	y_3	y_4
6	x_1	0	6	9	6
4	x_2	0	4	5	4
5	x_3	5	0	2	0
6	x_4	6	4	0	0

6 zeros

first revised cost matrix



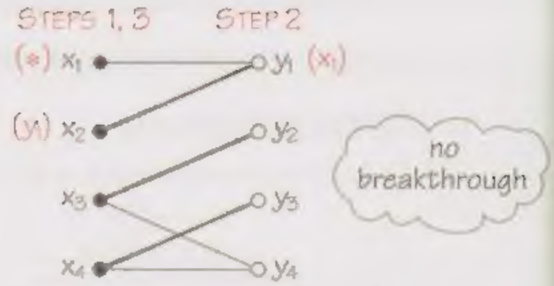
first partial graph

2 SOLUTION CONTINUED

PART A: LABEL VERTICES

		0	0	0	3
		y_1	y_2	y_3	y_4
6	x_1	0	6	9	6
4	x_2	0	4	5	4
5	x_3	5	0	2	0
6	x_4	6	4	0	0

first revised cost matrix



labelled partial graph

PART C: MODIFY PARTIAL GRAPH

STEP 6: FIND δ

unlabelled ys

		0	0	0	3
		y_1	y_2	y_3	y_4
6	x_1	0	6	9	6
4	x_2	0	4	5	4
5	x_3	5	0	2	0
6	x_4	6	4	0	0

labelled xs

first revised cost matrix

unlabelled ys

	y_2	y_3	y_4
x_1	6	9	6
x_2	4	5	4

labelled xs

δ = 4

STEP 7: REVISE COST MATRIX AND PARTIAL GRAPH

2 decrease by δ

		−4	0	0	3
		y_1	y_2	y_3	y_4
10	x_1	0	2	5	2
8	x_2	0	0	1	0
5	x_3	9	0	2	0
6	x_4	10	4	0	0

3 decrease by δ

1 increase by δ

4 increase by δ

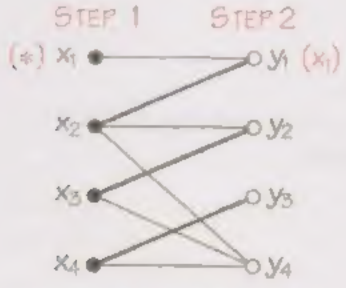
second revised cost matrix

second partial graph

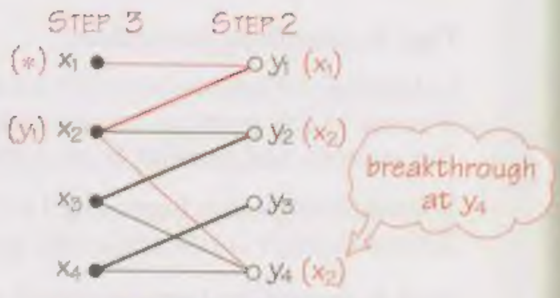
TRACK 18

2 SOLUTION CONTINUED

PART A: LABEL VERTICES



second partial graph

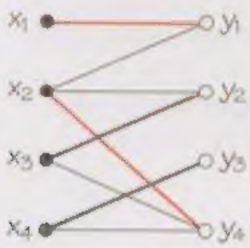


labelled partial graph

PART B: IMPROVE MATCHING

STEP 4: an alternating path is $y_4x_2y_1x_1$.

STEP 5: revised matching is:



new matching
x_3y_2, x_4y_3, x_1y_1, x_2y_4

PART A: LABEL VERTICES (impossible) STOP

	y_1	y_2	y_3	y_4
x_1	6	12	15	15
x_2	4	8	9	11
x_3	10	5	7	8
x_4	12	10	6	9

original cost matrix

		-4	0	0	3
		y_1	y_2	y_3	y_4
10	x_1	0	2	5	2
8	x_2	0	0	1	0
5	x_3	9	0	2	0
6	x_4	10	4	0	0

final revised cost matrix

optimum assignment: x_1y_1, x_2y_4, x_3y_2, x_4y_3

total cost: $6 + 11 + 5 + 6 = 28 = 10 + 8 + 5 + 6 - 4 + 0 + 0 + 3$

TRACK 19

3 SUMMARY OF THE ALGORITHM

START with no matching.

Assign weights to the vertices and construct the first partial graph.

Part A: labelling procedure

Label the vertices to identify an alternating path.
If none of the vertices on one side of the graph can be labelled, STOP:
the current assignment is an optimum assignment.

If *breakthrough* is achieved, go to Part B.
If *breakthrough* is not achieved, go to Part C.

SHORT CUT
first time only – find
matching by inspection

Part B: matching improvement procedure

Find an alternating path by tracing back through the labels.
Form a new matching.
Return to Part A.

Part C: modification of the partial graph procedure

Construct a revised cost matrix as follows.

On the existing cost matrix:
draw a *horizontal* line through each labelled vertex in X;
draw a *vertical* line through each labelled vertex in Y;

- find the smallest entry δ with ONLY a *horizontal* line through it;
- *decrease* all entries with ONLY a *horizontal* line through them by δ; *increase* the weight on the corresponding vertices in X by δ;
- *increase* all entries with ONLY a *vertical* line through them by δ; *decrease* the weight on the corresponding vertices in Y by δ.

Construct a revised partial graph.
(Remove any edge that now has a non-zero cost.)

Return to Part A.

TRACK 20

4 PROBLEM

Find the optimum assignment in the following case.

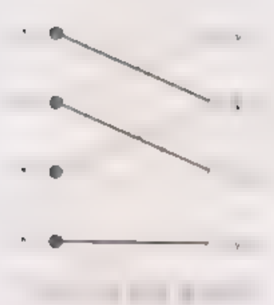
	y_1	y_2	y_3	y_4
x_1	8	4	6	7
x_2	10	12	8	14
x_3	9	6	11	15
x_4	12	8	14	8

cost matrix

TRACK 21

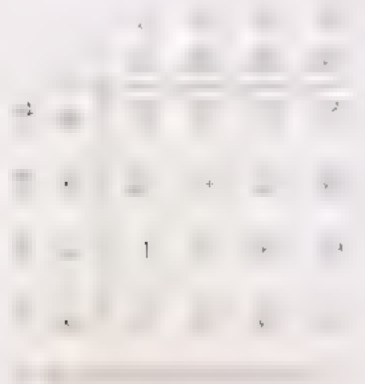
5 SOLUTION

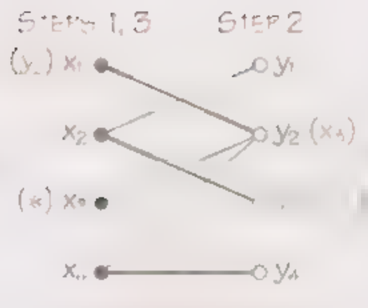
STEP 0: CONSTRUCT INITIAL PARTIAL GRAPH



5 SOLUTION CONTINUED

PART A: LABEL VERTICES

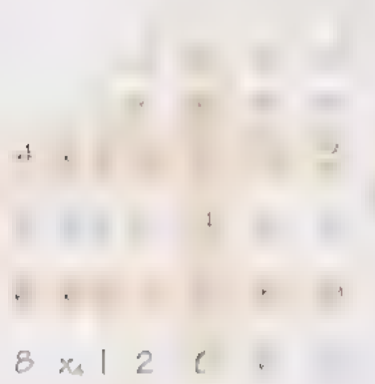




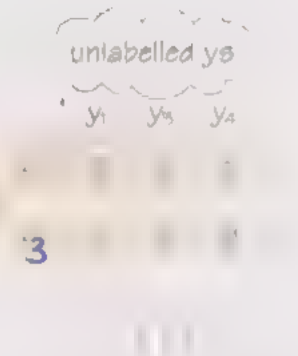
labelled partial graph

PART C: MODIFY PARTIAL GRAPH

STEP 6: FIND δ



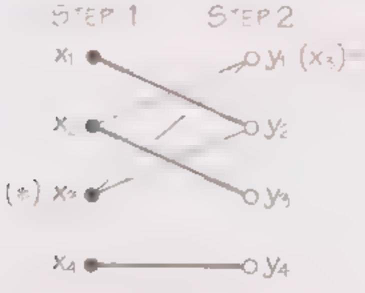
first revised cost matrix



STEP 7: REVISE COST MATRIX AND PARTIAL GRAPH

5 SOLUTION CONTINUED

PART A: LABEL VERTICES



abelled partial graph

PART B: IMPROVE MATCHING

STEP 4: an alternating path is y_1x_3

STEP 5: revised matching is:

new matching
x_1y_2, x_2y_3, x_4y_4, x_3y_1

PART A: LABEL VERTICES impossible, STOP

	y_1	y_2	y_3	y_4
x_1	[illegible]	[illegible]	[illegible]	[illegible]
x_2	[illegible]	[illegible]	[illegible]	[illegible]
x_3	[illegible]	[illegible]	1	[illegible]
x_4	[illegible]	[illegible]	11	[illegible]

origina cost matrix

	y_1	y_2	y_3	y_4
x_1	[illegible]	[illegible]	1	[illegible]
x_2	[illegible]	[illegible]	0	[illegible]
x_3	[illegible]	[illegible]	4	[illegible]
x_4	2	1	6	[illegible]

final revised cost matrix

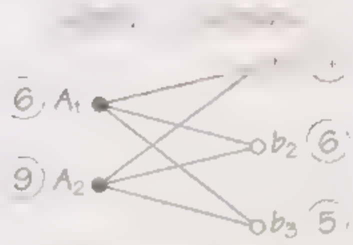
optimum assignment: x_1y_2, x_2y_3, x_3y_1, x_4y_4

tota cost: $4 + 8 + 9 + 8 = 29 = 5 + 8 + 7 + 8 + 2 - 1 + 0 + 0$

The Hungarian algorithm for the transportation problem

USE
- labelling procedure
- flow-augmenting procedure
- modification of partial graph procedure

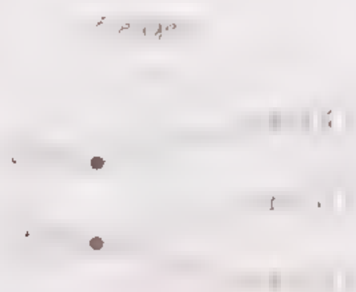
1 WORKED PROBLEM



bipartite graph

2 SOLUTION TO WORKED PROBLEM

STEP 0: CONSTRUCT INITIAL PARTIAL GRAPH


first partial graph

TRACK 25

2 SOLUTION CONTINUED

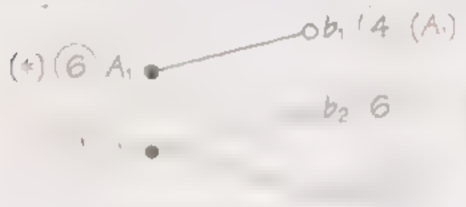
PART A: LABEL VERTICES

STEPS 1–3

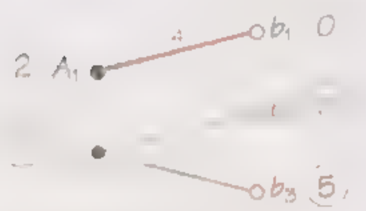
PART B: AUGMENT FLOW

STEPS 4, 5

First iteration

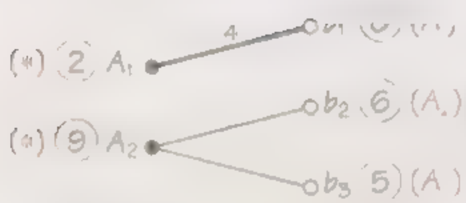


breakthrough at b_1

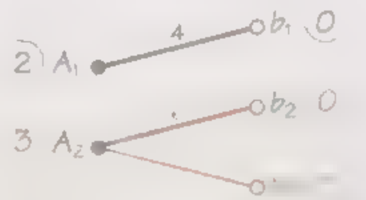


- flow-augmenting path is A_1b_1
- min (6, 4) = 4
- send flow of 4 down A_1b_1

Second iteration

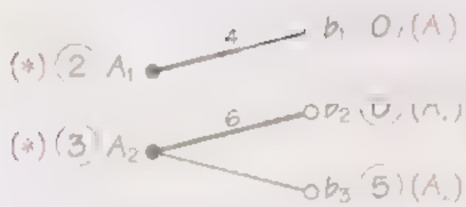


breakthrough at b_2



- flow-augmenting path is A_2b_2
- min (9, 6) = 6
- send flow of 6 down A_2b_2

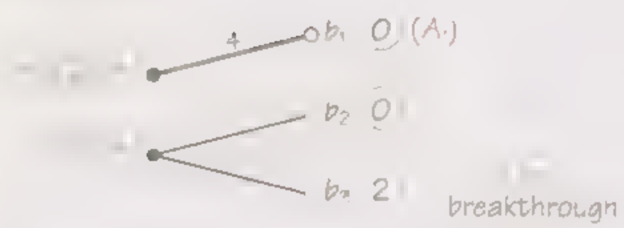
Third iteration



breakthrough at b_3

- flow-augmenting path is A_2b_3
- min (3, 5) = 3
- send flow of 3 down A_2b_3

Fourth iteration

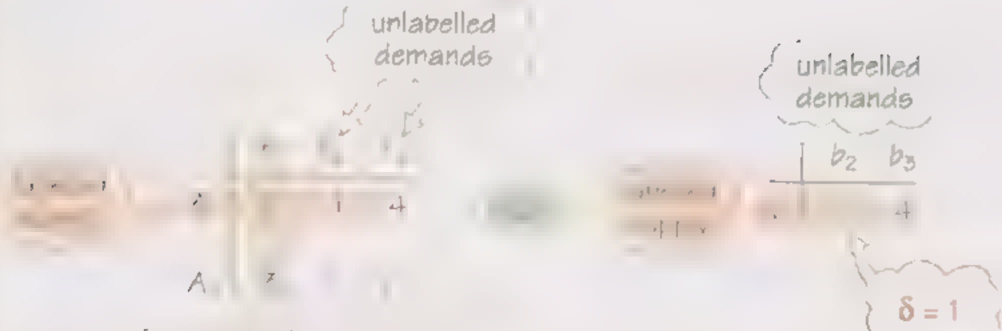


TRACK 23

2 SOLUTION CONTINUED

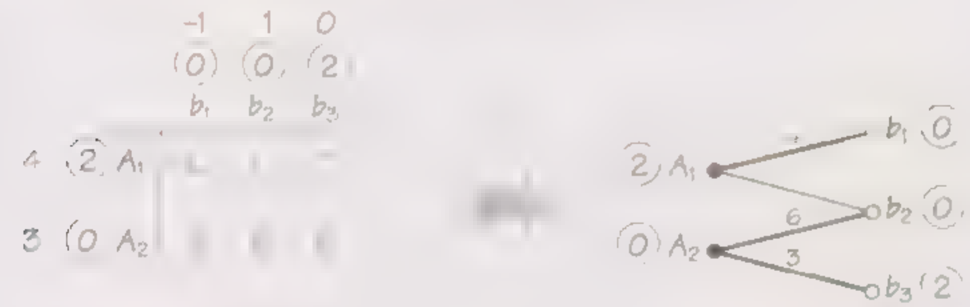
PART C: MODIFY PARTIAL GRAPH

STEP 6: FIND δ



first revised cost matrix

STEP 7: REVISE COST MATRIX AND PARTIAL GRAPH

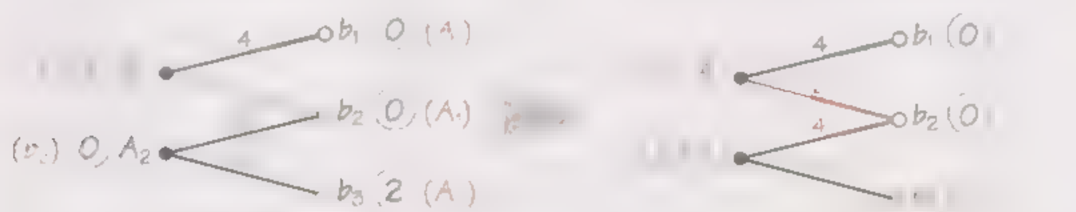


second revised cost matrix

second partial graph

PART A: LABEL VERTICES

PART B: AUGMENT FLOW



breakthrough at b_3

maximum flow along A_2b_3 is 6

- flow-augmenting path is $b_3A_2b_2A_1$
- min (6, 2, 2) = 2
- increase flow by 2 on A_1b_2, A_2b_3
- decrease flow by 2 on A_2b_2

final flow is 4 along A_1b_1 with cost (4 × 3)
2 along A_1b_2
4 along A_2b_2
5 along A_2b_3 + (5 × 3) = 53

3 SUMMARY OF THE ALGORITHM

START with no flow.

Construct the initial partial graph.

Part A: labelling procedure

Label the vertices to identify a flow-augmenting path.
If no labelling is possible, STOP:
the current solution is a minimum-cost solution

If *breakthrough* is achieved, go to Part B
If *breakthrough* is not achieved, go to Part C

Part B: flow-augmenting procedure

Find a flow-augmenting path by tracing back through the labels
Augment the flow.
Return to Part A.

Part C: modification of the partial graph procedure

Construct a new revised cost matrix as follows.

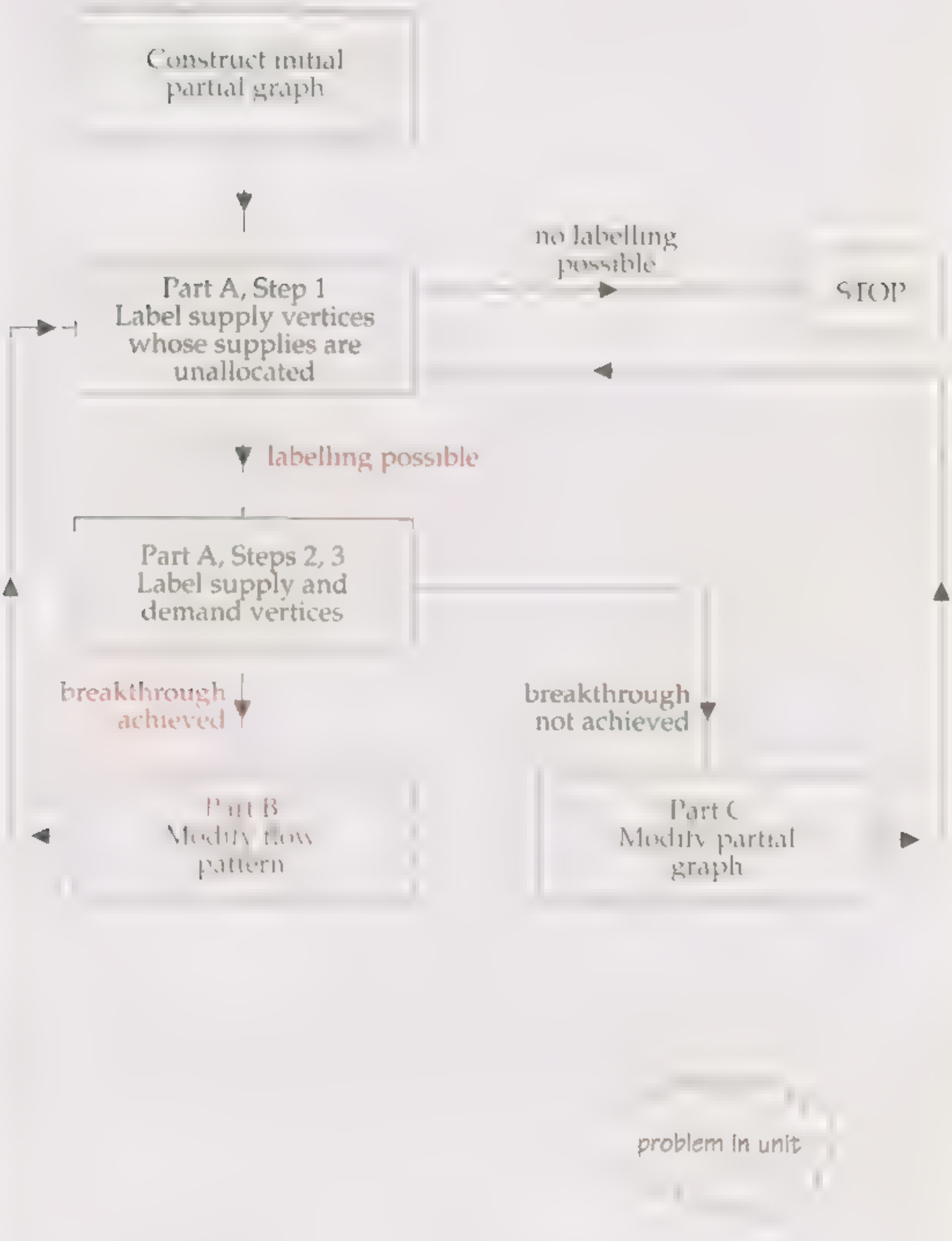
On the existing cost matrix:
draw a *horizontal* line through each labelled *supply* vertex;
draw a *vertical* line through each labelled *demand* vertex.

- find the smallest entry δ with only a *horizontal* line through it
- *decrease* all entries with only a *horizontal* line through them by δ
  increase the weight on the corresponding *supply* vertices by δ
- *increase* all entries with only a *vertical* line through them by δ
  decrease the weight on the corresponding *demand* vertices by δ

Construct a revised partial graph.
(Remove any edge that now has a non-zero cost.)

Return to Part A.

4 FLOW CHART FOR THE ALGORITHM



An algorithm for the knapsack problem

1 WORKED PROBLEM

Consider five items with the following weights and values.

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

USE
- branching tree
- lower bounds

[illegible]

2 SOLUTION VECTORS

A **solution vector** is a sequence of the form $(x_1, x_2, x_3, x_4, x_5)$,

$$\text{where} \begin{cases} x_i = 1 & \text{if item } i \text{ is packed;} \\ x_i = 0 & \text{if item } i \text{ is not packed.} \end{cases}$$

A **feasible solution** is one which satisfies the weight constraint.

(1, 1, 0, 0, 1) corresponds to items A, B and E packed, with total weight $w = 4 + 2 + 1 = 7$ (≤ 9) ✓ — feasible solution

(0, 1, 1, 0, 1) corresponds to items B, C and E packed, with total weight $w = 2 + 7 + 1 = 10$ (> 9) ✗ — infeasible solution

3 BRANCHING IDEA

For example:

[illegible] → (0, 1, 1, 0, 0)
[illegible] → (0, 1, 0, 1, 0)
[illegible] → (0, 1, 0, 0, 1)

add a 1 to the right of this 1

these have one more item than [illegible] solution vector

4 DECIDING HOW TO BRANCH

For example:

(1, 0, 0, 0, 0)

- (1, 1, 0, 0, 0) $w = 6$
- (1, 0, 1, 0, 0) $w = 11$ ✗ infeasible
- (1, 0, 0, 1, 0)
- (1, 0, 0, 0, 1) w

branching possible

Next step: branch out from (1, 1, 0, 0, 0).

5 OUTLINE OF ALGORITHM

START with zero vector (0, 0, …, 0); feasible with value 0.
STORE (0, 0, …, 0) and value 0.

GENERAL STEP

- Branch from first solution from which branching is possible.
- Calculate total weight of each new solution.
- Calculate total value of each new *feasible* solution.
  If there is a new feasible solution with value greater than the value stored, store the new solution vector and its value instead.
- Mark vertex with □ if it corresponds to:
  - □ a vector which equals or exceeds weight restriction;
  - □ a vector which ends in 1.

REPEAT the GENERAL STEP until no more branching is possible.

STOP Stored solution vector and value is optimum solution.

6 SOLUTION TO WORKED PROBLEM

[illegible]	A	[illegible]	[illegible]	[illegible]	E
[illegible]	4	[illegible]	[illegible]	[illegible]	1
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1

[illegible] not more than 9

First branching: from zero solution vector $\boldsymbol{O} = (0, 0, 0, 0, 0)$ with $v = 0$:

O •
- • (1, 0, 0, 0, 0) [illegible]
- • (0, 1, 0, 0, 0) [illegible]
- • (0, 0, 1, 0, 0) [illegible]
- • (0, 0, 0, 1, 0) [illegible]
- • (0, 0, 0, 0, 1) [illegible] 1 [illegible] 1

STORE (0, 0, 1, 0, 0), $v = 9$

Second branching: from (1, 0, 0, 0, 0):

O •
- • (1, 0, 0, 0, 0)
  - • [illegible] 11
  - • (1, 0, 1, 0, 0) [illegible] ✕
  - • (1, 0, 0, 1, 0) [illegible]
  - • (1, 0, 0, 0, 1) [illegible] 1
- • (0, 1, 0, 0, 0)
- • (0, 0, 1, 0, 0)
- • (0, 0, 0, 1, 0)

STORE (1, 1, 0, 0, 0), $v = 11$

Third branching: from (1, 1, 0, 0, 0):

O •
- • (1, 0, 0, 0, 0) —— • (1, 1, 0, 0, 0)
  - • (1, 1, 1, 0, 0) $w = 13$ ✕
  - • (1, 1, 0, 1, 0) $w = 11$ ✕
  - • (1, 1, 0, 0, 1) $w = 7$ $v = 12$
- • (0, 1, 0, 0, 0)
- • (0, 0, 1, 0, 0)
- • (0, 0, 0, 1, 0)

STORE (1, 1, 0, 0, 1), $v = 12$

6 SOLUTION CONTINUED

Fourth branching: from (0, 1, 0, 0, 0):

O •
• (0, 1, 0, 0, 0)
• (0, 0, 1, 0, 0)
• (0, 0, 0, 1, 0)

• (0, 1, 1, 0, 0) $w = 9$ $v = 17$
• (0, 1, 0, 1, 0) $w = 7$ $v = 14$
$\boxed{\bullet}$ (0, 1, 0, 0, 1) $w = 3$ $v = 9$

STORE (0, 1, 1, 0, 0), $v = 17$

Fifth branching: from (0, 1, 0, 1, 0):

O •
• (0, 1, 0, 0, 0) —— • (0, 1, 0, 1, 0) —— $\boxed{\bullet}$ (0, 1, 0, 1, 1) $w = 8$ $v = 15$
•
$v = 6$ • (0, 0, 0, 1, 0)

STORE unchanged

Sixth branching: from (0, 0, 1, 0, 0):

O •
• (0, 0, 1, 0, 0)
$v = 6$ • (0, 0, 0, 1, 0)

• (0, 0, 1, 1, 0) $w = 12$ ✗
• (0, 0, 1, 0, 1) $w = 8$ $v = 10$

STORE unchanged

Seventh branching: from (0, 0, 0, 1, 0):

O • —— • (0, 0, 0, 1, 0) —— $\boxed{\bullet}$ (0, 0, 0, 1, 1) $w = 6$ $v = 7$

No further branching is possible: STOP.

Solution vector is (0, 1, 1, 0, 0): items *B* and *C*, with value 17.

An algorithm for the travelling salesman problem

1 WORKED PROBLEM

Find a 5-cycle through A, B, C, D, E with minimum total length:

	A	B	C	D	E
A		[illegible]	[illegible]	[illegible]	[illegible]
B	[illegible]		[illegible]	[illegible]	[illegible]
C	[illegible]	[illegible]		[illegible]	[illegible]
D	[illegible]	[illegible]	[illegible]		[illegible]
E	[illegible]	[illegible]	[illegible]	[illegible]	

USE

- lower bounds
- branching tree

2 GETTING LOWER BOUND FROM TABLE

	A	B	C	D	E
A		[illegible]	[illegible]	[illegible]	[illegible]
B	[illegible]		[illegible]	[illegible]	[illegible]
C	1	[illegible]		[illegible]	[illegible]
D	[illegible]	[illegible]	[illegible]		[illegible]
E	[illegible]	[illegible]	[illegible]	[illegible]	

need

- one entry from each row
- one entry from each column
- no 2-, 3- or 4-cycles

	A	B	C	D	E
A		[illegible]	[illegible]	[illegible]	[illegible]
B	[illegible]		[illegible]	[illegible]	[illegible]
C	[illegible]	[illegible]		[illegible]	[illegible]
D	[illegible]	[illegible]	[illegible]		[illegible]
E	[illegible]	[illegible]	[illegible]	[illegible]	

lower bound is $1 + 2 + 1 + 1 + 1 = 6$

	A	B	C	D	E
A		[illegible]	[illegible]	[illegible]	[illegible]
B	[illegible]		[illegible]	[illegible]	[illegible]
C	[illegible]	[illegible]		[illegible]	[illegible]
D	[illegible]	[illegible]	[illegible]		1
E	[illegible]	[illegible]	[illegible]	1	

new lower bound is $6 + 1 = 7$

3 DECIDING HOW TO BRANCH

Consider edges with zero weight:

(try to get maximum increase in lower bound)

	A	B	C	D	E
A					
B					
C					
D					
E					

exclude AC?

	A	B	C	D	E
A			X		
B					
C					
D					
E					

	A	B	C	D	E
A					
B					
C					
D					
E					

		A	B	C	D	E
	A					
1	B				X	
	C					
	D					
	E					

lower bound increases by 1 + 0 = 1

Label each zero with possible increase in lower bound.

Select an edge whose exclusion gives maximum increase in lower bound.

	A	B		D	E
A					
B					
C					
D					
E					

maximum increase in lower bound arises from excluding AC

4 CARRYING OUT BRANCHING

	A	B	C	D	E
A	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
B	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
C	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
D	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
E	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

select AC

lower bound 7

include AC
(so exclude CA)

	A	B	D	E
B	[illegible]	[illegible]	[illegible]	[illegible]
C	X	[illegible]	[illegible]	[illegible]
D	[illegible]	[illegible]	[illegible]	[illegible]
E	[illegible]	[illegible]	[illegible]	[illegible]

exclude AC (cross out AC)

	A	B	C	D	E
A	[illegible]	[illegible]	X	[illegible]	[illegible]
B	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
C	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
D	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
E	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

5 OUTLINE OF ALGORITHM

START with a given $n \times n$ table of distances, corresponding to a complete weighted graph with n vertices.

Carry out the initial row and column reduction, and calculate the initial lower bound.

GENERAL STEP

- Consider all the edges with zero weight, and choose an allowable edge e whose *exclusion* leads to the *greatest increase* in lower bound; if there are several such edges, choose the first.
- Consider the consequences of including e and excluding e. Use row and column reduction to determine these consequences, in terms of increases in the lower bound. Choose the option which gives the *smaller* lower bound; if the lower bounds are equal, *include* the edge e. STORE the current list of included edges, and the current lower bound.
- Continue from the current position *unless* the chosen option has a lower bound greater than a previously eliminated option, in which case backtrack to the earlier position.

REPEAT the GENERAL STEP until a cycle with n edges has been created

STOP Stored list of edges is optimum solution.

6 SOLUTION TO WORKED PROBLEM: FIRST BRANCHING

Cons der edges with zero weight:

	A	B	C	D	E
A					
B					
C					
D					
E					

nclude AC

	A	B	D	E
B				
C	X			
D				
E				

exclude AC

	A	B	C	D	E
A			X		
B					
C					
D					
F					

reduce column E by 1

	0	0	0	1
	A	B	D	E
B				
C	X			
D				
E				

reduce column C by 2

	0	0	2	0	0
	A	B	C	D	E
A			X	0	0
B			0	0	
C				4	
D					
E		1		4	

7 SECOND BRANCHING

Consider edges with zero weight:

3	A	B	C	D	E
A	–	[illegible]	X		
B	[illegible]	–			[illegible]
C	[illegible]	[illegible]		4	[illegible]
D	[illegible]		[illegible]	–	4
E	0[1]	1	3	4	–

select BC
(lower bound increases by 2, the maximum possible)

include BC
(so exclude CB)

4	A	B	D	E
A	[illegible]	[illegible]	[illegible]	[illegible]
C		X	4	[illegible]
D		[illegible]		4
E	[illegible]	1	4	

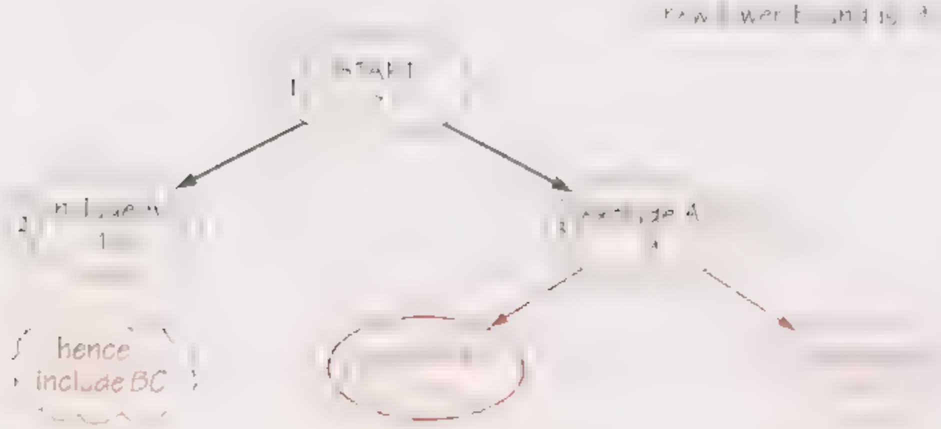
lower bound remains 9

exclude BC

	A	B	C	D	E
A		[illegible]	X	[illegible]	[illegible]
B	[illegible]		X	[illegible]	
C	[illegible]	[illegible]		4	[illegible]
D	[illegible]	[illegible]	[illegible]		[illegible]
E			[illegible]	4	

reduce column C by 2

5	A	B	C	D	E
A		[illegible]	X	0	[illegible]
B	[illegible]		X	0	1
C	[illegible]	[illegible]		4	[illegible]
D	[illegible]	[illegible]	[illegible]		[illegible]
E		1	1	4	



TRACK 14

8 THIRD BRANCHING

Consider edges with zero weight:

4	A	B	D	E
A		[illegible]		
C		X	4	[illegible]
D				4
E		1	1	

select AD
(lower bound increases by 4, the maximum possible)

include AD
(so exclude DA)

	A	B	E
C		X	
D	X		4
E		1	

exclude AD

	A	B	D	E
A		[illegible]	X	0
C	[illegible]	X	4	[illegible]
D	[illegible]	[illegible]		4
E		1	4	

[illegible]

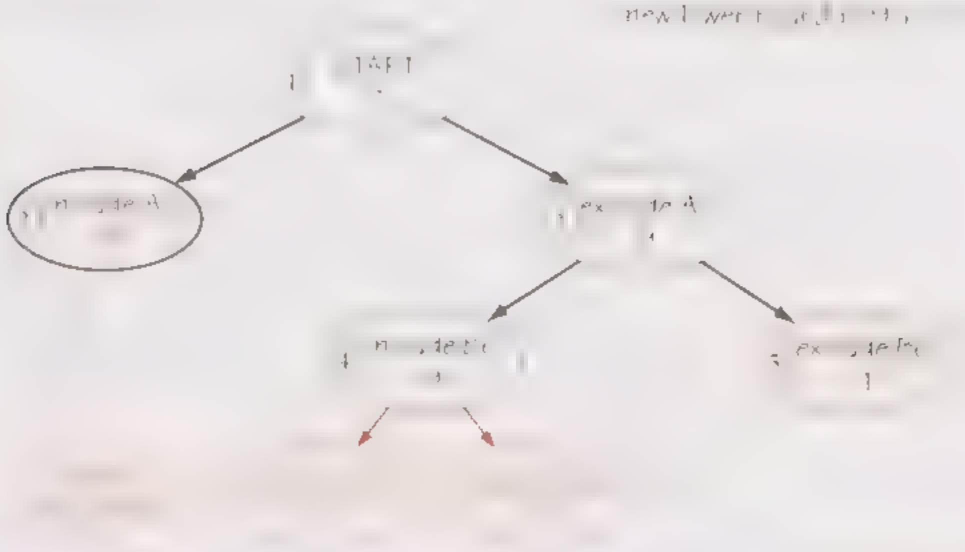
[illegible]

[illegible]	A	B	E
C		X	[illegible]
D	X		
E			

[illegible]

7	A	B	D	E
A		[illegible]	X	
C		X		[illegible]
D	[illegible]			4
E		1		

[illegible]



9 FOURTH BRANCHING

Consider edges with zero weight:

2	A	B	D	E
B				
C	X			
D				
E				

select BD
(lower bound increases by 2, the maximum possible)

include BD
(so exclude DB)

	A	B	E
C	X		
D		X	
E			

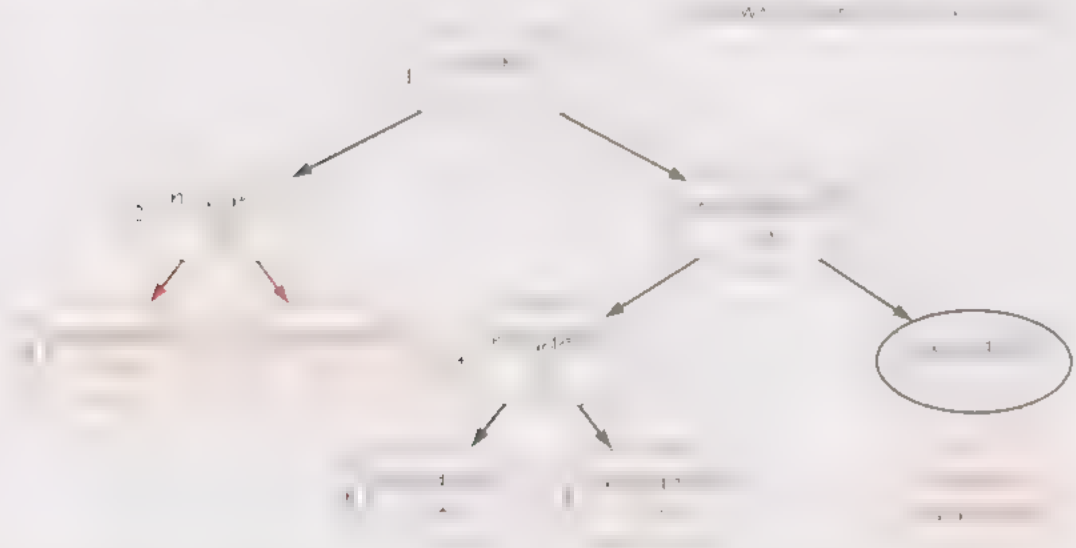
exclude BD

	A	B	D	E
B			X	
C	X			
D				
E				

reduce column E by 2

	0	0	2
x	**A**	**B**	**E**
C	X		
D		X	
E			

4	A	B	D	E
B			X	
C	X			
D				
E		1		



TRACK 16

10 FIFTH BRANCHING

Consider edges with zero weight:

5	A	B	C	D	E
A	–	6	X	0^0	0^1
B	6	–	X	0^1	1
C	0^2	2	–	4	5
D	0^0	0^1	0^1	–	4
E	0^1	1	1	4	–

select CA
(lower bound increases by 2, the maximum possible)

include CA

	B	C	D	E
A	6	X	0	0
B	–	X	0	1
D	0	0	–	4
E	1	1	4	–

reduce row E by 1

	10	B	C	D	E
0	A	6	X	0	0
0	B	–	X	0	1
0	D	0	0	–	4
1	E	0	0	3	–

new lower bound is 11 + 1 = 12

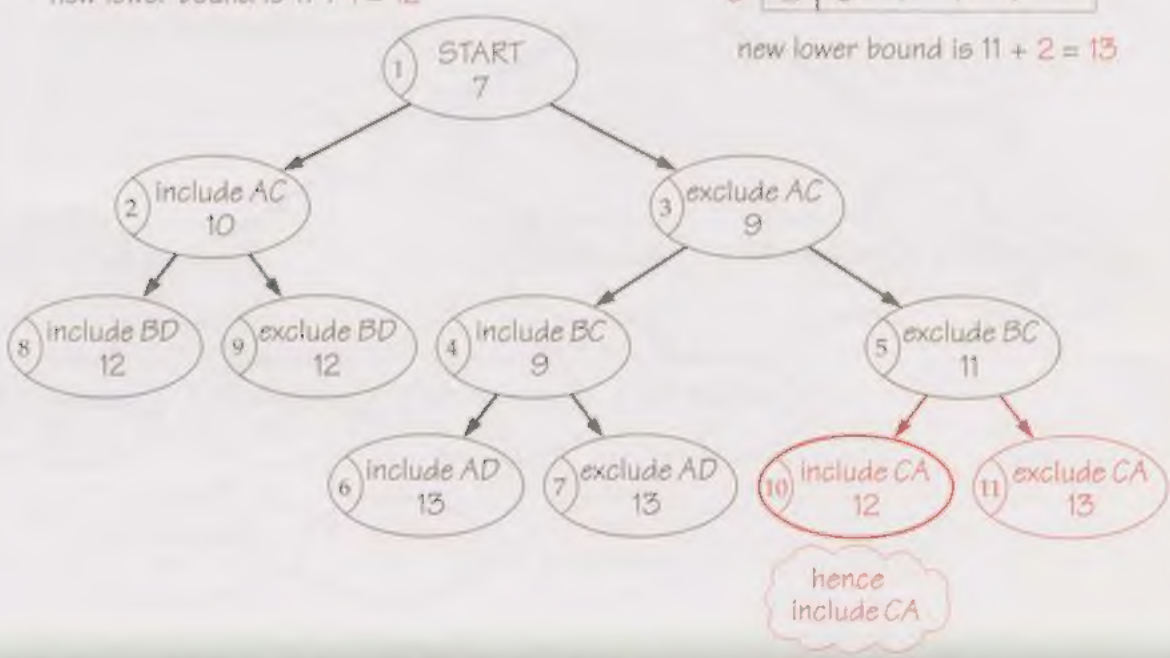
exclude CA

	A	B	C	D	E
A	–	6	X	0	0
B	6	–	X	0	1
C	X	2	–	4	5
D	0	0	0	–	4
E	0	1	1	4	–

reduce row C by 2

	11	A	B	C	D	E
0	A	–	6	X	0	0
0	B	6	–	X	0	1
2	C	X	0	–	2	3
0	D	0	0	0	–	4
0	E	0	1	1	4	–

new lower bound is 11 + 2 = 13



TRACK 17

11 SIXTH BRANCHING

Consider edges with zero weight:

10	B	C	D	E
A	6	X	0^0	0^1
B	–	X	0^1	1
D	0^0	0^0	–	4
E	0^0	0^0	3	–

select AE
(lower bound increases by 1, the maximum possible)

include AE
(so exclude EC to avoid 3-cycle AECA)

12	B	C	D
B	–	X	0
D	0	0	–
E	0	X	3

lower bound remains 12

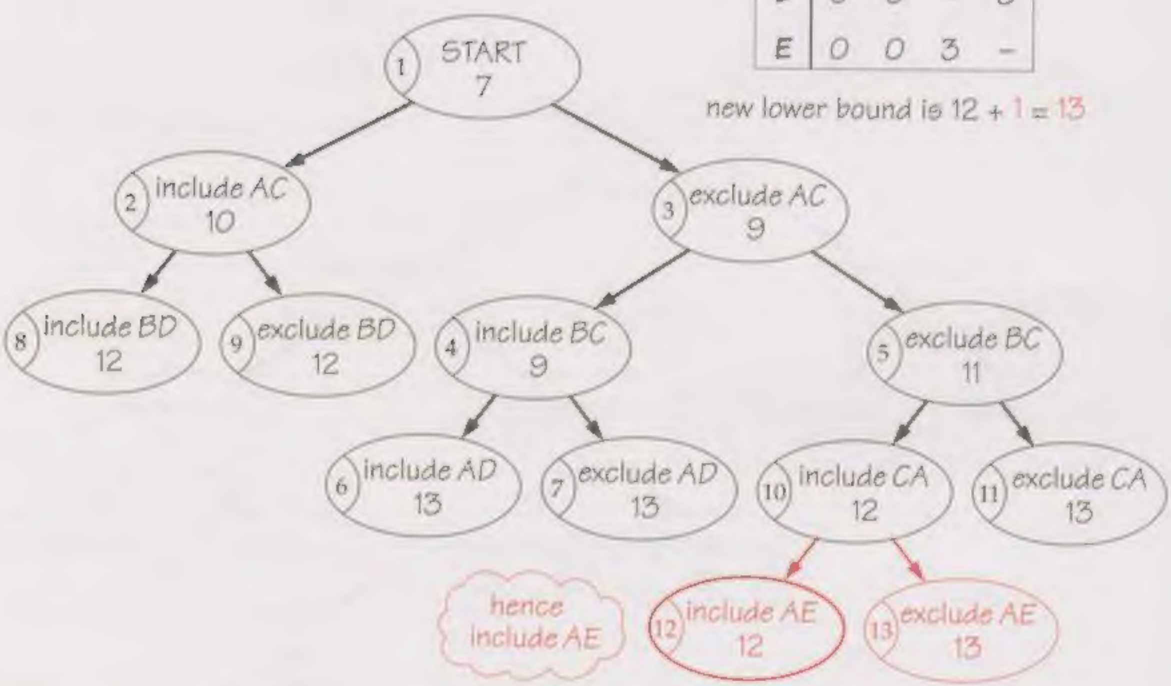
exclude AE

	B	C	D	E
A	6	X	0	X
B	–	X	0	1
D	0	0	–	4
E	0	0	3	–

reduce column E by 1

	0	0	0	1
13	**B**	**C**	**D**	**E**
A	6	X	0	X
B	–	X	0	0
D	0	0	–	3
E	0	0	3	–

new lower bound is 12 + 1 = 13



TRACK 18

12 FINAL BRANCHINGS

Consider edges with zero weight:

12	B	C	D
B	–	X	0^3
D	0^0	0^0	–
E	0^3	X	3

select BD
(lower bound increases by 3, the maximum possible)

include BD
(so exclude DB)

14	B	C
D	X	0
E	0	X

lower bound remains 12

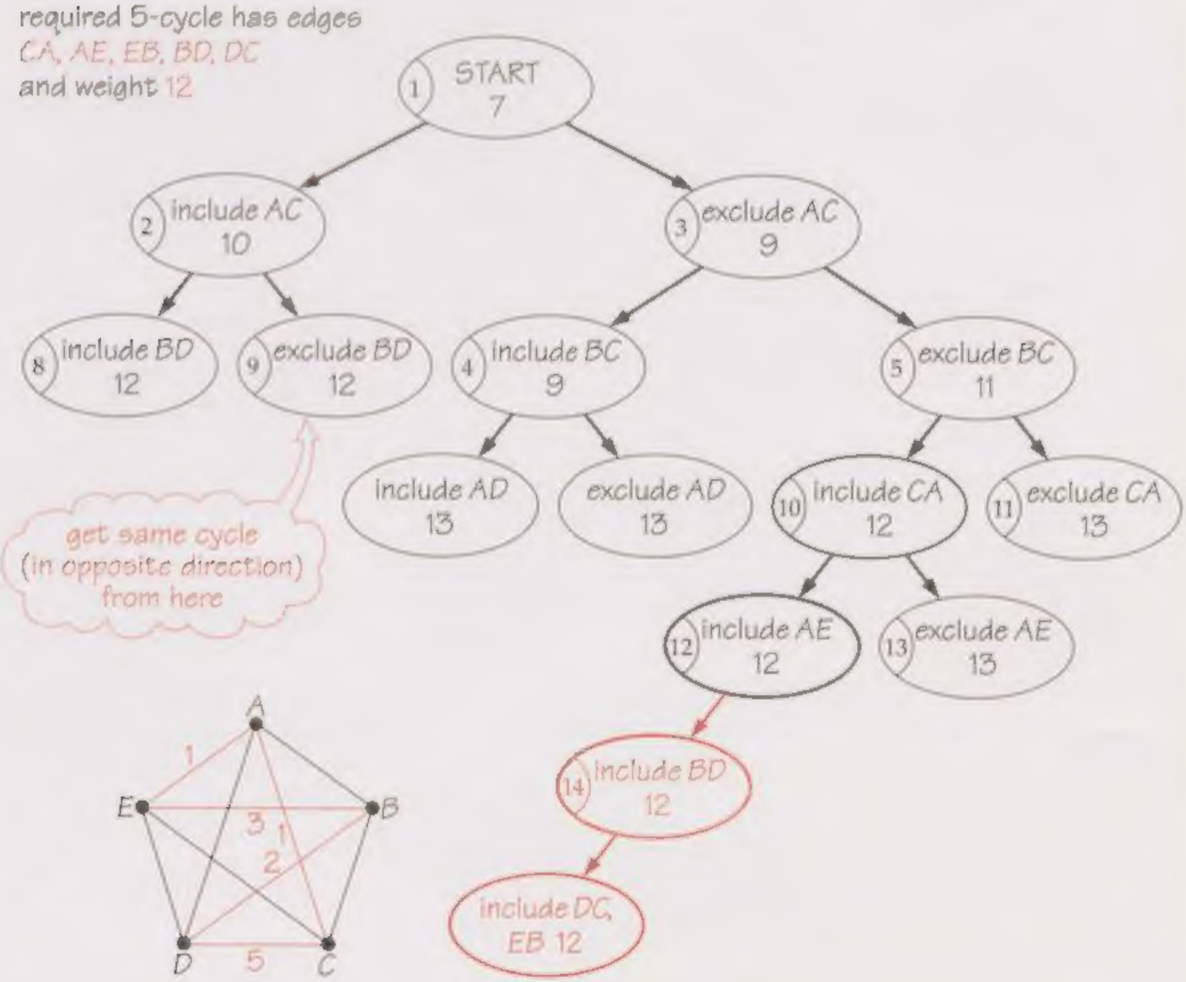
include DC and EB
lower bound remains 12

exclude BD

15	B	C	D
B	–	X	X
D	0	0	–
E	0	X	3

not possible!
(no route out of B)

required 5-cycle has edges
CA, AE, EB, BD, DC
and weight 12



Printed in the United Kingdom.